Name:

ID:

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 5 – Due Thurs Feb 20 11:55pm **Spring 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Informal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solutions:

- All submissions must be easily legible.
- You should submit your work through the **class Canvas page** only.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please allot at least as many pages per problem (or subproblem) as are allotted in this template.

Quicklinks: 1a 1b 2a 2b 3a 3b 3c

---

1. *In this question we consider the change-making problem (as covered in recitation), of making change for n cents using the smallest number of coins. Suppose we have coins with denominations of $v_1 > v_2 > \cdots > v_r$ for r coins types, where each coin's value v_i is a positive integer. Your goal is to obtain a set of counts $\{d_i\}$, one for each coin type, such that $\sum_{i=1}^{r} d_i = k$ and where k is minimized, and such that the sum of the values $\sum_{i=1}^{r} d_i v_i = n$.*

   (a) *A greedy algorithm for making change is the **cashier's algorithm**. Consider the following pseudocode—meant to implement the cashier's algorithm—where n is the amount of money to make change for and v is a vector of the coin denominations:*

```
change(n,v,r) :
   d[1 .. r] = 1       // initial histogram of coin types in solution
   while n > 0 {
      k = r
      while ( k > 0 and v[k] > n ) { k++ }
      if k==0 { return 'no solution' }
      else { n = n - v[k] }
   }
   return d
```

This code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

There are 3 *logical* errors in his algorithm. One of these logical errors has resulted in several problems in the code, all of which need to be changed.

First, the coins are indexed in the wrong order; the greedy choice is to choose the largest coin denomination that fits within n, but the code goes in the reverse order, starting with the smallest denomination v_r. This error alone would cause an otherwise correct cashier's algorithm to return only a count of the smallest denomination coins. Fixing this error requires changing several things in the code; the initial coin index `k = r` should be `k = 1`, the loop-termination condition `k > 0` should be `k <= r`, and finally the *no solution* condition `k==0` should be `k == r+1`.

Second, the histogram that contains the count of which coin values to return is incorrectly initialized, so that the initial "solution" is to take 1 of each kind of coin. This is fixed by setting `d[1..r] = 0` (instead of 1).

Third, the histogram never gets incremented. This bug will cause the algorithm to return the `d` array as initialized, with 1 (or 0, once the above bug is fixed) of each coin denomination, since none of the $d[k]$ values are changing. To correct this, we simply add the line `d[k]++` within the `else` branch of the `if` conditional.

The corrected algorithm is

```
change(n,v,r) :
   d[1 .. r] = 0 // initial histogram of coin types in solution
   while n > 0 {
      k = 1
      while ( k <= r and v[k] > n ) { k++ }
      if k == r+1 { return 'no solution' }
      else {
         n = n - v[k]
         d[k]++
      }
   }
   return d
```

(b) *Identify a set of Euro coin denominations (a subset of the denominations here)*[1] *for which the greedy algorithm does not yield an optimal solution for making change. Justify your answer in terms of optimal substructure and the greedy-choice property. (The set should include the 1 Euro cent so that there is a solution for every value of n.) Include an example where the cashier's algorithm with your choice of denominations yields a set of coins that is larger than it needs to be, and also show the smaller set of coins adding up to the same value.*

For $\{1, 20, 50\}$ cents, $60 = 50$ cents$+10\times$1-cent in greedy approach. However, the optimal approach is $60 = 20$ cents$\times 3$.

This problem *does* have optimal substructure, since given the optimal solution to (certain) subproblems (in this example, $n = 40$), one can reconstruct the optimal solution to the original problem However, the greedy-choice property does not hold. For greedy choice, first choosing 50 cents and then 10 1-cent does not achieve global optimum. We need to think globally (need to refer to future decisions and other possible solutions) to get a optimal solution (choose three 20-cent coins).

[1] `https://www.google.com/search?q=euro+coin+denominations`